RL-TR-96-56
Final Technical Report
April 1996

ROBUST TRANSFIGURING NETWORK PROTOCOLS

SRI International

Richard G. Ogier, Julie S. Wong, and Irfan H. Khan

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Rome, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be releasable to the general public, including foreign nations.

RL-TR- 96-56 has been reviewed and is approved for publication.

APPROVED: Charles Meyer

CHARLES MEYER
Project Engineer

FOR THE COMMANDER: John A Graniero

JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1996	Final Mar 92 - Mar 95

4. TITLE AND SUBTITLE

ROBUST TRANSFIGURING NETWORK PROTOCOLS

5. FUNDING NUMBERS

C - F30602-92-C-0052
PE - 62702F
PR - 4519
TA - 22
WU - 22

6. AUTHOR(S)

Richard G. Ogier, Julie S. Wong, and Irfan H. Khan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

SRI International
383 Ravenswood Avenue
Menlo Park CA 94025-3493

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Rome Laboratory/C3BC
525 Brooks Rd
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

RL-TR-96-56

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Charles Meyer/C3BC/(315) 330-1880

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

In RTNP, we have developed a protocol that uses two artificial intelligence (AI) methods, neural networks and evidential reasoning, to recognize and predict adverse network conditions, and that uses fuzzy logic to dynamically control the parameters of a tunable routing protocol in response to the perceived environment. Examples of the tunable protocol parameters are: 1) a parameter that controls the degree to which traffic is spread over multiple paths; 2) a link bias parameter that, when large, increases stability by forcing traffic over minimum-hop paths; and 3) a parameter that determines how often routing updates are sent. Examples of measurements used to recognize adverse conditions are: 1) congestion; 2) probability of a successful transmission on a link; 3) jamming characteristics; and 4) degree of routing oscillations.

Neural network methods were developed for predicting link-states and congestion, based on network measurements and estimates. These methods were shown in simulations to predict link states and queuing delay much more accurately than other methods.

14. SUBJECT TERMS

Artificial intelligence, Evidential reasoning, Protocols, Neural networks, Algorithms, Networks

15. NUMBER OF PAGES

40

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

SRI International (SRI) is pleased to submit this final technical report for the Robust Transfiguring Network Protocols (RTNP) Project, sponsored by Rome Laboratory under Contract No. F30602-92-C-0052.

1.1 OBJECTIVE

U.S. Air Force (USAF) C3I networks must be able to function efficiently despite very stressful conditions that may include a highly dynamic topology due to mobility and jamming, sophisticated electronic countermeasures (ECM) attacks, and unpredictable, dynamic traffic demand. Programs such as the Evaluation and Development of Multimedia Networks in Dynamic Stress (EDMUNDS) program [Hight et al. 1993] have produced network protocols that permit effective service in such stressful, dynamic environments. However, each such protocol was designed to operate effectively under a particular model of the network environment. It is not sufficient to select a single protocol that achieves optimal performance in the most severe possible environment, since this selection may result in poor performance in other environments. Therefore, different protocols are required under different environmental conditions.

The purpose of RTNP was therefore to develop methods to recognize or predict adverse network conditions (having either malicious or benign causes) that may result in poor protocol performance, and to dynamically select the most appropriate protocol or protocol parameters in response to the recognized conditions.

Unfortunately, in an operational mode, knowledge of the environment is likely to be uncertain, incomplete, and erroneous, especially when the environment changes rapidly, and particularly when the changes are due to sophisticated ECM attacks. For example, it may be difficult to ascertain the jamming characteristics or the sophistication of the adversary.

Therefore, to solve the above problem, two subproblems need to be solved:
(1) to recognize environmental conditions that cause a given protocol to fail (we define a protocol as failing if another known protocol would perform significantly better in the same situation); and (2) to decide which protocol or protocol parameter values to use based on the perceived environment.

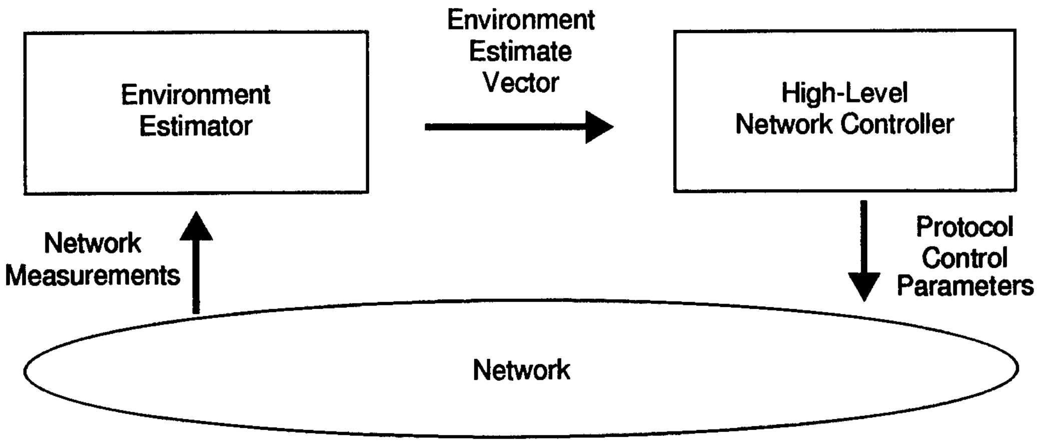
Environment Estimator
Environment Estimate Vector
High-Level Network Controller
Network Measurements
Protocol Control Parameters
Network

Figure 1.1. RTNP Concept

1.2 METHOD OF APPROACH

The RTNP solution to this problem is to combine robust tunable protocols with methods that continually gather network measurements, analyze those measurements to recognize environmental conditions, and dynamically select the most appropriate protocol and/or protocol parameter values to optimize performance in the perceived environment. This solution combines conventional network protocols for low-level network control with AI methods for environment estimation and high-level network control.

The general concept of RTNP is shown in Figure 1.1. The environment estimator (EE) obtains network state information and updates the environment estimate vector (EEV), which represents its view of the environment. The high-level network controller (HLNC) reads the EEV (and in some cases performance feedback information), and selects the parameters that determine which member of the protocol family to use.

The RTNP project is divided into four tasks. The roles and interdependence of these four tasks can be summarized as follows (see Figure 1.2):

Task 1—Exploration of Protocol Failures. Using simulations, gain an understanding of which protocols (or protocol parameters) should be selected if the environment were known exactly, and how the performance degrades if the wrong protocol is chosen for a given environment. Determine which environmental characteristics have the most effect on the protocol selection; these will be the characteristics that need to be estimated using knowledge-based techniques in Task 2.

Task 2—Artificial Intelligence Environment Estimation. Design a knowledge-based system that will maintain and update an estimate of the environment. The environment estimate vector will provide possibly uncertain and incomplete information about a carefully selected set of environmental characteristics that have the

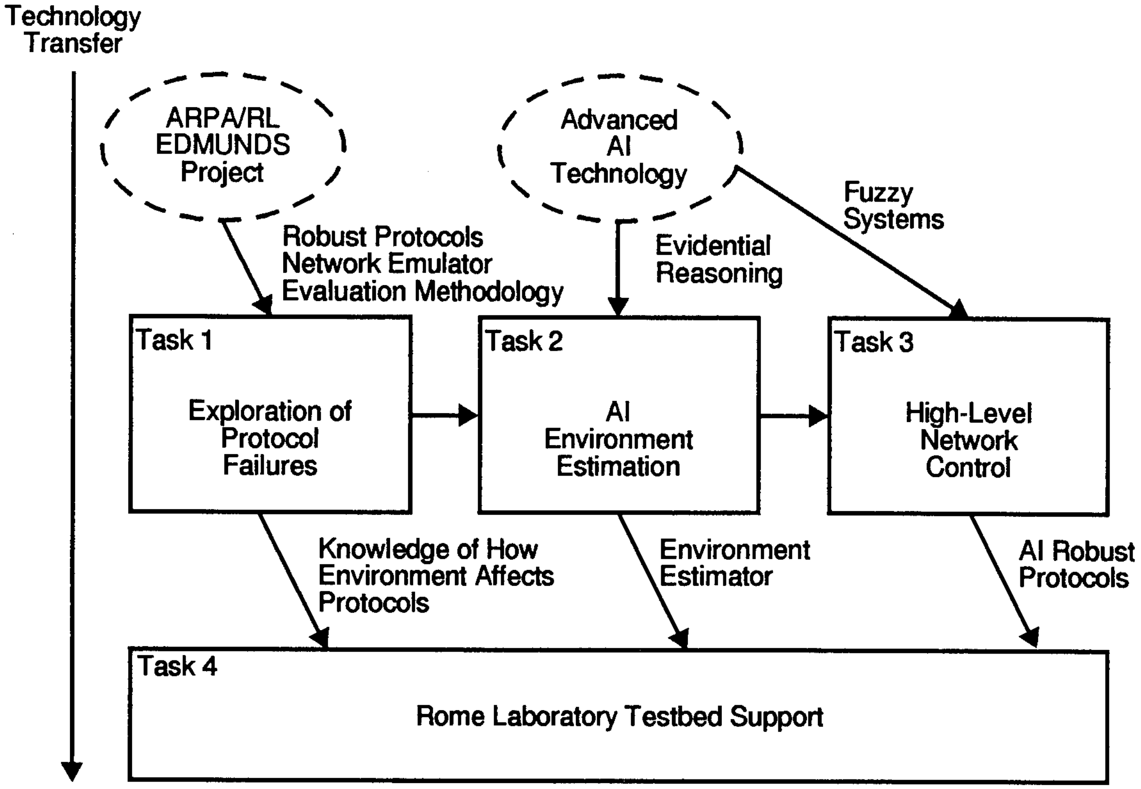

Figure 1.2. RTNP Task and Program Relationships

potential of causing protocol failures.

Task 3—High-Level Network Control. Design the HLNC, which will read the EEV and select the appropriate protocol or protocol parameters. The HLNC performs the function F:X → P, where X is the space of EEVs, P is the space of protocol parameter vectors, and F(x) is the vector consisting of the chosen parameters for EEV x.

Task 4—Rome Laboratory Support. Provide support to RL to assist in in-house testing and demonstration of the protocol technology developed during the proposed effort. This support will include the installation of network emulation capabilities, demonstration experiment scripts, and on-site support.

1.3 SUMMARY OF ACCOMPLISHMENTS

In RTNP we have developed a protocol that uses two artificial intelligence (AI) methods, neural networks and evidential reasoning, to recognize and predict adverse network conditions, and that uses fuzzy logic to dynamically control the parameters of a tunable routing protocol in response to the perceived environment. The Secure Tactical Internet Protocol 3 (STIP3) [Hight et al. 1993] developed in EDMUNDS was selected as the routing protocol, because of its proven effectiveness and its large number of tunable parameters. However, the methods developed under RTNP can be adapted to any tunable protocol.

Examples of the tunable protocol parameters are (1) a parameter that controls the degree to which traffic is spread over multiple paths, (2) a link bias parameter that, when large, increases stability by forcing traffic over minimum-hop paths, and (3) a parameter that determines how often routing updates are sent. Examples of measurements used to recognize adverse conditions are (1) congestion, (2) probability of a successful transmission on a link, (3) jamming characteristics, and (4) degree of routing oscillations.

The main accomplishments of RTNP can be summarized as follows:

Neural Network Methods for Network State Prediction. Neural network methods were developed for predicting link-states and congestion, based on network measurements and estimates. These methods were shown in simulations to predict link states and queuing delay much more accurately than other methods.

Fuzzy Logic HLNC. An HLNC based on fuzzy logic was developed that dynamically modifies the STIP3 protocol parameters in response to changes in network measurements and estimates. Fuzzy logic rules were found that resulted in better performance than STIP3 in some scenarios. However, we found it difficult to find fuzzy rules that consistently perform better than STIP3 in all scenarios.

The Application of Gister to Environment Estimation. Gister is an artificial intelligence tool developed at SRI that applies the technique of evidential reasoning to draw conclusions about hypotheses, given evidence from different sources. An EE module based on Gister was designed that estimates environment characteristics based on network measurements. In addition, we extended Gister (which normally produces LISP code) to produce C code, so that the code can be integrated into the RTNP software. The effectiveness of Gister depends heavily on the problem being solved and on the design of the Gister system. Our testing of Gister was limited, but simulations showed neural networks to be superior to Gister for predicting link states.

RTNP Architecture and Protocol. We developed the RTNP architecture and protocol, which integrates neural networks, evidential reasoning, and fuzzy logic, to recognize and predict adverse network conditions and dynamically select the most appropriate parameter values for the underlying network protocol (STIP3).

The RTNP simulation software was developed by modifying and augmenting the software already developed in EDMUNDS for simulating STIP3. Both the STIP3 and RTNP software are written in C++, using the Network Algorithm Programmer's Interface (NAPI+), also developed in EDMUNDS. The RTNP software was designed to be flexible, to allow future experimentation. For example, the user can

- Specify the neural network connection weights in an input file.
- Specify the fuzzy rules for the HLNC in an input file.
- Specify that the EE and HLNC not be used, so that RTNP becomes STIP3.

- Specify the Gister analysis, either using the Gister tool or by writing a new C function.
- Replace STIP3 with another underlying network protocol.

1.4 ORGANIZATION OF REPORT

The remainder of this report is organized as follows. Section 2 provides an overview of the RTNP protocol, Section 3 gives an overview of the RTNP simulation software, Section 4 summarizes the simulation test results, and Section 5 presents conclusions. References are listed in Section 6.

This report is intentionally concise. Additional details on the material summarized in this report can be found in the following documents:

- *Software Design Document for the Robust Transfiguring Network Protocol* [Khan, Ogier, and Wong 1995a], which specifies the RTNP protocol
- *Software Test Report for the Robust Transfiguring Network Protocol* [Khan, Ogier, and Wong 1995b], which describes the tests performed for RTNP and presents their results
- *Software User's Manual for the Robust Transfiguring Network Protocol* [Wong, Ogier, and Khan 1995], which gives the procedures for using the RTNP simulation software.

2 OVERVIEW OF RTNP

This section presents an overview of the RTNP protocol. A detailed description of RTNP can be found in the Software Design Document [Khan, Ogier, and Wong 1995a]. This section discusses the protocol independently of the software that we used to simulate it. A discussion of the RTNP simulation software is given in Section 3, and instructions on using the software are given in the Software User's Manual [Wong, Ogier, and Khan 1995].

The architecture for RTNP at a single network node is shown in Figure 2.1. The protocol is identical at each node. The underlying network protocol is STIP3, which was developed in the EDMUNDS project. We present a brief overview of STIP3 below. A detailed description of STIP3 is given in Hight et al. [1993]. RTNP was developed by adding new modules for environment estimation and high-level network control to STIP3. These new modules allow various STIP3 control parameters to be controlled dynamically in response to changes in the perceived environment.

RTNP is a distributed protocol; that is, the processing is done at each node rather than by a centralized processor, and information is exchanged between neighboring nodes. A distributed architecture was chosen for the following reasons:

- A distributed architecture is more survivable than a centralized one.
- A centralized architecture requires a large amount of bandwidth to transport network-state information from all nodes to the central processor.
- A centralized architecture results in slower response to environment changes, because of the communication delays between the nodes and the central processor.

Referring to Figure 2.1, the measurements module collects local measurements and estimates received by neighbors, and forwards them to the two EE modules and to the HLNC module. The EE modules use these measurements to estimate relevant environment characteristics, such as the amount of jamming. These estimates are then forwarded to the HLNC, which uses them to control the STIP3 control parameters based on fuzzy logic. Some of these estimates—in particular, predictions for link-state probabilities and queuing delays—are also forwarded directly to the STIP3 module, which may use them in place of the less accurate estimates normally used by STIP3 for computing the routing variables.

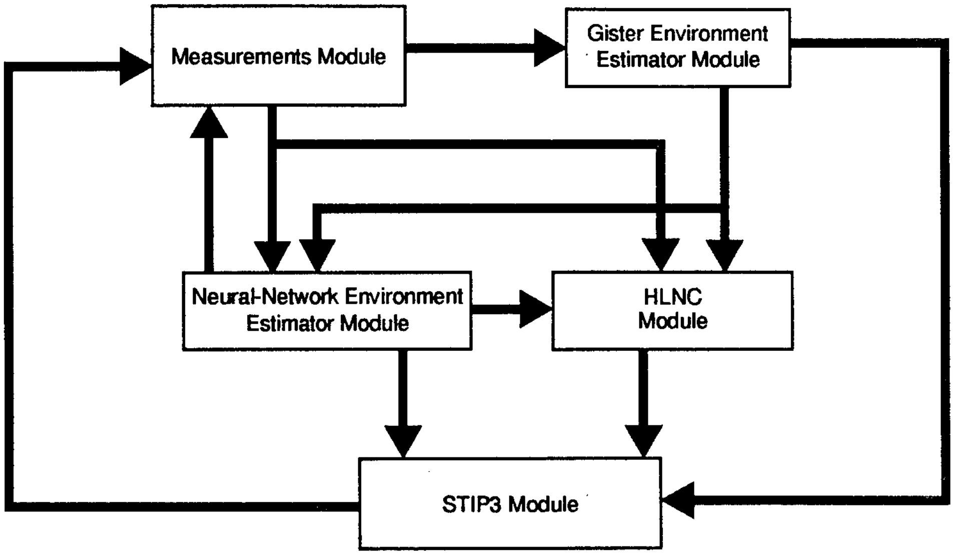
Measurements Module
Gister Environment Estimator Module
Neural-Network Environment Estimator Module
HLNC Module
STIP3 Module

Figure 2.1. RTNP Modules and Interfaces

The EE is divided into two modules: the Gister EE Module and the Neural-Network EE Module. Gister is an AI software tool developed at SRI that applies the technique of evidential reasoning to draw conclusions about hypotheses, given evidence from different sources. As discussed below, neural networks and evidential reasoning have different advantages and disadvantages, and therefore complement each other. The RTNP architecture is flexible in that a given implementation of RTNP may use either, both, or neither of these two modules. This allows, for example, the two methods to be compared.

The architecture is also flexible in that a given implementation of RTNP may or may not use the HLNC. If the HLNC and both EE modules are omitted, then RTNP becomes STIP3. This allows RTNP to be compared with STIP3 without requiring separate software for STIP3. In addition, it may be desirable to use only STIP3 in networks that have a predictable environment.

The fuzzy rules used by the HLNC, the neural-network connection weights used by the neural-network EE, and the Gister analysis, can be specified by the user. This flexibility allows the fuzzy rules and neural networks to be tailored to a particular network. It also allows different fuzzy rules, neural networks, and Gister analyses to be evaluated and compared. The neural-network connection weights are obtained by training the neural network off-line, as described in the the Software User's Manual [Wong, Ogier, and Khan 1995].

The following subsections discuss the functions of the RTNP modules.

2.1 STIP3 MODULE

This subsection presents a brief overview of the STIP3 protocol. STIP3 is a network protocol that performs dynamic multipath routing, link scheduling, and flow control. It is a very flexible protocol that has several control parameters that can be tuned to greatly change its character in order to adapt to changes in the environment. STIP3 was designed for highly dynamic networks that can suffer frequent changes in topology and link capacities due to mobility, interference, and jamming. It was designed for point-to-point links, but can be adapted to multiple access networks. STIP3 was shown in simulations to perform much better than previously existing routing protocols based on shortest paths and local queuing delay [Hight et al. 1993].

STIP3 is a distributed, distance-vector algorithm. Each node i maintains and updates the distance variables $e(i,d)$ for each destination d, where $e(i,d)$ represents the expected time for the last packet at node i and destined for node d to arrive at node d. The distance variables $e(i,d)$ are time smoothed and are sent to neighbors only if they change by at least some threshold amount. This time smoothing results in the distance information being successively averaged as it propagates across the network, so that the farther it is propagated, the more it is averaged.

STIP3 uses destination queues instead of link queues, so that the routing decision can be made at the last possible moment. In STIP3, each node computes link flows $f(l,d)$ for each link l and destination d, which determine the percentage of packets with destination d that are to be routed on link l. These flows are computed so as to minimize the sum of $e'(i,d)$ over d, where $e'(i,d)$ is the unsmoothed expected delay to d. (Alternatively, the sum of the squared delays can be minimized for improved fairness.) This flow optimization takes into account the expected delays received from neighbors, the capacities of the outgoing links, the estimated reliability of each outgoing link, the current local queue sizes, and measurements of the arrival traffic. These values are used to predict the near-future average queuing delay, which is used to compute $e'(i,d)$.

The receiver of a successful transmission sends an acknowledgment (ack) to the transmitter; if no ack is received within a timeout window, the packet is queued for retransmission. Since STIP3 uses destination queues instead of link queues, a packet need not be retransmitted on the same link on which it was previously transmitted. This is desirable since the previous link may have deteriorated since the last transmission.

The reliability $p(l)$ of each link (i.e., the probability that a packet transmission on the link will be successful) is updated dynamically based on the history of transmissions and acks. This algorithm exploits the following inherent asymmetry: if an ack has been received for a transmission, then we know the transmission was successful, but if an ack has not yet been received, we do not know whether the transmission was successful. Based on this asymmetry, the algorithm computes two different estimates for $p(l)$, one of which is a lower bound, and then sets $p(l)$ to the maximum of these estimates. This technique results in faster response to a link coming up than the simple averaging method used in STIP2. In addition, STIP3 adjusts the smoothing constant for $p(l)$ depending on the rate of link dynamics, since the optimal smoothing constant was found to depend greatly on this rate.

To minimize routing loops that could be caused by network dynamics, STIP3 uses a distributed algorithm to compute and update a directed acyclic graph, DAG(d), for each destination d, based on the expected delays to that destination. Packets with destination d are limited to links in DAG(d), and each node considers only links in the current DAG(d) when computing the flows $f(l, d)$.

STIP3 provides the option of using a simple flow control mechanism that stops admitting traffic destined for node d if $e(i, d)$ exceeds a given threshold.

STIP3 employs a token-based link-scheduling algorithm that determines the next packet to send on an available link. This scheduling algorithm has the following attributes: (1) traffic of a given destination is guaranteed to be divided among outgoing links in proportion to the computed flows $f(l, d)$; (2) traffic burstiness is taken into consideration, unlike the weighted round robin method; (3) destination queues are used, so that the routing decision can be made at the last possible moment; and (4) multiple priorities are supported.

In summary, STIP3 has the following attributes:

- Quickly finds alternate routes in response to jamming and congestion.
- Allocates link resources optimally, given the available information.
- Uses a new updating method that successively averages delay information as it is propagated.
- Works effectively for arbitrarily fast link dynamics.
- Queues packets at nodes, not links, so that routing decisions can be based on the most recent state and delay information for each link.
- Employs directed acyclic graphs to reduce routing loops.

Some limitations of STIP3 are as follows:

- Requires a large amount of processing per packet, limiting the number of packets that can be processed per second. This limitation can be overcome by using larger packets.
- Requires a large amount of processing to compute flows. This limitation can be overcome by using a more efficient flow-computation algorithm and/or computing flows less frequently.
- Employs datagram routing, and therefore does not preserve packet sequence.

- May suffer from transient routing loops, since it is based on distance vectors. This limitation can be overcome at the expense of increased message overhead by distributing link-state information.
- Does not reserve bandwidth or provide quality-of-service guarantees.

Interfaces Between STIP3 and the Other Modules. Some measurements already computed by STIP3, such as link-state probabilities, link jamming rates, and queue sizes, are used by the measurements module to compute other measurements. In the other direction, the link-state probabilities and queuing delays computed by the neural-network EE can optionally be used by STIP3 in place of the estimates normally used by STIP3 for computing the routing flows.

Any of the STIP3 control parameters can be controlled dynamically by the HLNC, based on the network measurements and the estimates computed by the two EE modules. Some of the parameters we controlled in experiments are as follows:

- *spread.* Determines the degree to which traffic of each destination is spread over multiple paths.
- *link_bias.* A constant which is added to the delay of each link. When *link_bias* is very large, STIP3 sends traffic alongs minimum-hop paths.
- *epsilon.* Determines the amount that a distance variable $e(i,d)$ must change before the new value is sent to neighbors.
- *alpha_inc.* Determines the amount of time smoothing for the distance variables $e(i,d)$.
- *short_update_period.* Determines the frequency at which update messages are sent to neighbors.

2.2 MEASUREMENTS MODULE

The Measurements Module collects and computes measurements that are used by the EE and HLNC modules. Some of these collected measurements are already computed by STIP3, and are used to compute other measurements or are forwarded to the other modules. Such measurements computed by STIP3 include the sizes $q(d)$ of the destination queues, the flows $f(l,d)$ on the outgoing links, the estimated link-state probabilities $p(l)$ and jamming rates $\lambda(l)$ for the outgoing links, counts of the arrival traffic, and a history of transmissions and acknowledgments.

Some measurements are received from neighboring nodes. These measurements, $ete\_jam\_amt\_nbr(l,d)$ and $ete\_jam\_rate\_nbr(l,d)$, measure the amount and rate of end-to-end jamming from each neighbor (corresponding to link l) to each destination d. They are used by the End-to-End Jamming Estimator Module to compute *ete_jam_amt*(d) and *ete_jam_rate*(d), which estimate the amount (degree) of jamming and the pulse rate of jamming that is present on the paths to each destination.

Several measurements are computed by the Measurements Module for use by the neural-network EE for predicting the link-state probabilities and queuing delays. These are discussed below in the subsection for the Neural-Network EE Module.

Several measurements are computed by the Measurements Module for use by the HLNC Module; these are discussed below in the subsection for the HLNC Module.

2.3 NEURAL-NETWORK EE MODULE

As discussed above, RTNP has two EE modules, one based on neural networks and the other based on Gister, since these two methods complement each other. Neural networks have the following advantages and disadvantages, as compared to Gister:

- Advantages
  - They are computationally faster than Gister.
  - They can learn any input-output function to arbitrary precision, given a large enough training set.
  - Unlike Gister, they do not require different sources of information to be independent.
- Disadvantages
  - They are difficult to analyze.
  - Their success depends on the training set being general enough.

The Neural-Network EE Module consists of the following submodules (as shown in Figure 2.2), the Link-State Predictor Module, the Queuing-Delay Predictor Module, and the End-to-End Jamming Estimator Module. The data output from these modules are estimates of the environment and may be used in the HLNC Module to control protocol parameters.

2.3.1 Link-State Predictor Module

This module is invoked at each epoch, for each link, to predict the probability that a packet transmitted on the link during the next epoch will be successful. We refer to this probability as the link-state probability.

This module implements an off-line-trained, feed-forward neural network, with an input layer, an optional hidden layer of neurons, and an output neuron [Haykin 1994; Masters 1993]. Figure 2.3 shows a feed-forward neural network with a single hidden layer. The output of the neural network gives the predicted link-state probability.

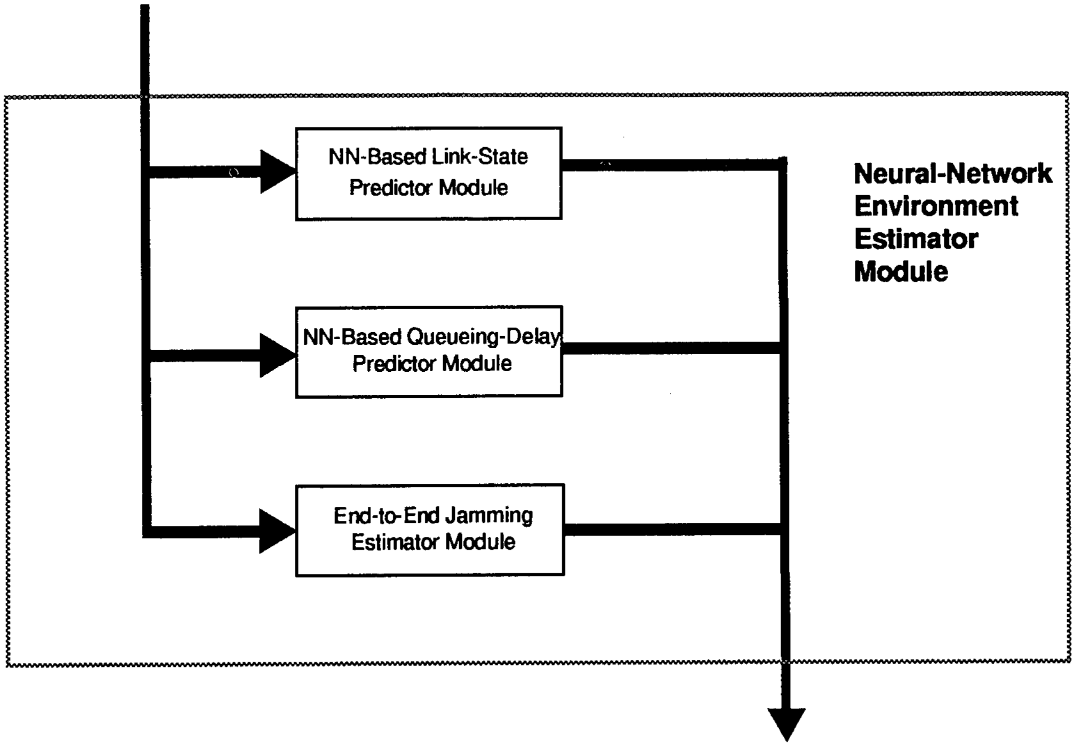
NN-Based Link-State Predictor Module
NN-Based Queueing-Delay Predictor Module
End-to-End Jamming Estimator Module
Neural-Network Environment Estimator Module

Figure 2.2. Neural-Network EE Submodules

For a given set of inputs $u_1, ..., u_I$, the output of the nerual network is given by

$$Y = g_{out}(\sum_{j=1}^{J} W_j V_j + b_{out}) \tag{2.1}$$

where V_j, $j = 1, ..., J$, is the output of the jth hidden neuron, given by

$$V_j = g(\sum_{i=1}^{I} w_{ij} u_i + b_j) \ . \tag{2.2}$$

The function $g(x)$ is the activation function for the hidden neurons, which we take to be $\tanh(x)$, the hyperbolic tangent function. The $\tanh(x)$ function is a sigmoid function that approaches $+1$ as x goes to ∞ and -1 as x goes to $-\infty$, and crosses zero at $x = 0$. The function $g_{out}(x)$ is the activation function for the output neuron, which we take to be either $\tanh(x)$ or the identity function $g(x) = x$. The W_j are the connection weights from the hidden neurons to the output neuron, the w_{ij} are those from the inputs to the hidden neurons, and b_{out} and b_j are the biases for the output neuron and hidden neurons, respectively.

The neural network is trained off-line with respect to a given training set, which consists of a representative set of input-output pairs. A procedure for training the neural network is provided in the Software User's Manual [Wong, Ogier, and Khan

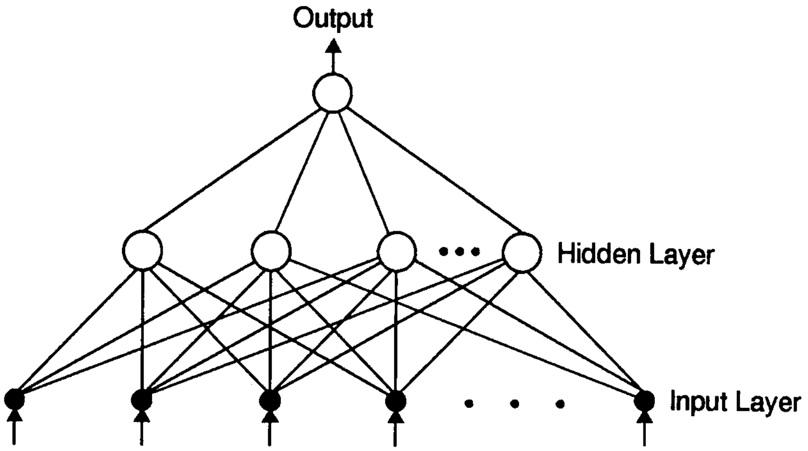

Figure 2.3. Feed-Forward Neural Network

1995]. This training gives the connection weights w_{ij} and W_j and the biases b_{out} and b_j, which are provided to RTNP via an input file.

The following variables, from the Measurements Module, can serve as inputs to the neural network, where $i = 0, 1, 2, \ldots, \lceil \frac{ack\_life}{\Delta t} \rceil$:

- $\lambda(l)$, the measured jamming rate on link l.
- *xmitted*(l, i) and *acked*(l, i), the number of packets transmitted on link l during the ith previous epoch, and the number of these packets that have been acknowledged (acked).
- *ackdelay_cdf*(l, i), the cumulative probability distribution function of the ack delay, in epochs.
- *avg_ackdelay*(l), the average ack delay on link l.
- *support*(l, i) = *acked*(l, i)/*xmitted*(l, i), which represents a lower bound on the probability of success on link l during the ith previous epoch.
- *plausibility*$(i, l) = 1 - [(1 - support(l, i))ackdelay\_cdf(l, i)]$, which represents an upper bound on the same probability.
- *avg_success_rate*$(l, 1)$ and *avg_success_rate*$(l, 2)$, exponentially decayed time averages, with two different time constants, of the fraction of transmissions that have been acked. (Using exponential time averages, we give more importance to recent transmissions.)

2.3.2 Queuing-Delay Predictor Module

The Queuing-Delay Predictor Module is invoked at each epoch processing instant, to predict the queuing delay at each queue averaged over the near future, i.e., over the next *qd_future* epochs, where *qd_future* is a parameter. (There is a separate queue at each node for each destination.)

As with the Link-State Predictor Module, this module implements a feed-forward neural network that is trained off-line and that has an input layer, an optional hidden layer of neurons, and an output neuron. The output of the neural network gives the predicted queuing delay.

The neural network is trained off-line with respect to a given training set, which consists of a representative set of input-output pairs. This training gives the neural network connection weights and biases, which are provided to RTNP via an input file.

The following variables, from the Measurements Module, can serve as inputs to the neural network:

- $q(d)$, the instantaneous queuing delay at the queue for destination d.
- $arr(d,i)$, $i = 0, 1, 2, \ldots, max\_history - 1$, the number of bits of destination-d traffic that arrived during the ith previous epoch.
- $avg\_arr(d,1)$, $avg\_arr(d,2)$ and $avg\_arr(d,3)$, exponentially decayed time averages, using short-, medium-, and long-term time constants, respectively, of $arr(d,i)$.
- *burstiness(d)*, a measure of the burstiness of the arrival traffic for destination d.
- $avg\_\lambda(d)$, the average jamming rate of the outgoing links, weighted by the link flows $f(l,d)$ for destination d.
- $total\_f(d)$, the sum of the link flows $f(l,d)$ over all outgoing links l.
- $p(d,1)$ and $p(d,2)$, averages over the outgoing links (weighted by the link flows for destination d) of medium- and long-term exponential time averages of the predicted link-state probabilities.

2.3.3 End-to-End Jamming Estimator Module

The End-to-End Jamming Estimator Module is called at each epoch processing instant to update the estimate of the amount and rate of end-to-end jamming to each destination. These estimates may be used by the HLNC Module for controlling the STIP3 protocol parameters.

The amount of end-to-end jamming to destination d, $ete\_jam\_amt(d)$, is a number between 0 and 1, where 0 represents no jamming and 1 represents perfect jamming (i.e., no packets can reach the destination). The jamming rate $\lambda(l)$ for a link represents the number of jamming pulses per second, and the end-to-end jamming rate for destination d, *ete_jam_rate*(d), is an estimate of the average jamming rate on paths leading to d.

Depending on the value of the parameter *add_jam_param_to_update*, nodes may periodically receive estimates, via update packets, of the amount and rate of end-to-end jamming from each neighbor to each destination. The latest such estimates are stored in the variables *ete_jam_amt_nbr(l,d)* and *ete_jam_rate_nbr(l,d).* These variables are initialized to 0 and are updated in the measurements module if and when new values for the variables arrive in update packets.

Dynamic programming (Bellman-Ford-like) equations are used to update the variables *ete_jam_amt*(d) and *ete_jam_rate*(d), based on the local jamming measurements *avg_jam_amt*(l) and $\lambda(l)$ and the estimates received from neighbors. The variables *ete_jam_amt*(d) and *ete_jam_rate*(d) are also time smoothed. If the parameter *add_jam_param_to_update* is equal to 1, then these variables are sent to neighbors via update packets when their quantized versions change. In addition, these variables are averaged over all d to give summaries of the amount and rate of jamming for the node, denoted *ete_jamming_amt* and *ete_jamming_rate*. The variable *ete_jamming_period*, defined as 1/*ete_jamming_rate*, is also computed, for use by the HLNC Module.

The End-to-End Jamming Estimator Module has the following input variables:

- *ete_jam_amt_nbr*(l, d) and *ete_jam_rate_nbr*(l, d)
- *avg_jam_amt*(l)
- $\lambda(l)$
- $f(l, d)$, the optimal effective flow on link l for packets for destination d
- $p(l)$, the predicted link-state-probability for link l over the next epoch.

The module provides the following output variables:

- *ete_jam_amt*(d)
- *ete_jam_rate*(d)
- *ete_jamming_amt*
- *ete_jamming_period*.

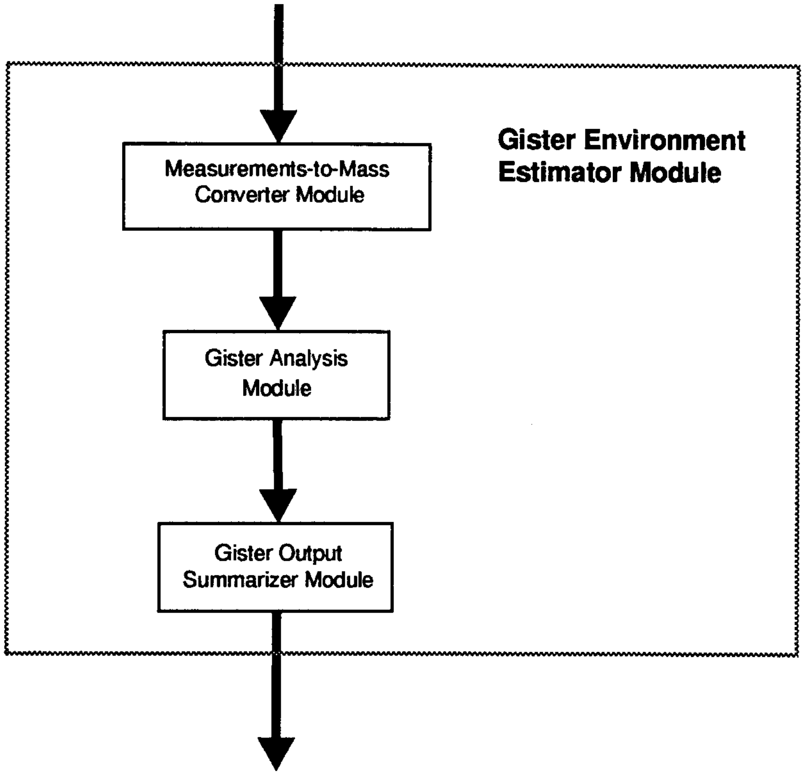
Gister Environment Estimator Module
Measurements-to-Mass Converter Module
Gister Analysis Module
Gister Output Summarizer Module

Figure 2.4. Gister EE Submodules

2.4 GISTER EE MODULE

The Gister EE Module uses Gister analysis (which is based on evidential reasoning) to obtain an estimate of the environment. The module consists of three submodules: the Measurements-to-Mass Converter Module, the Gister Analysis Module, and the Gister Output Summarizer Module, as shown in Figure 2.4. These modules are discussed briefly in the following subsections. Appendix B of the Software Design Document [Khan, Ogier, and Wong 1995a] contains a review of Gister and evidential reasoning. A more complete description of Gister can be found in the Gister manual [Lowrance 1993].

2.4.1 Measurements-to-Mass Converter Module

The Measurements-to-Mass Converter Module is invoked at each epoch processing instant to convert measurements from the measurements module into masses needed as inputs for Gister analysis. (In evidential reasoning, a *mass* is a degree of belief assigned to a proposition.)

Each measurement that is input to this module is associated with a Gister frame that contains a number of propositions associated with the measurement. For example, if the measurement is the amount of jamming, then the propositions of the associated frame might be *small, medium, and large.*

In addition, each measurement is associated with a fuzzy domain (see Subsection 2.5), which contains a fuzzy set for each proposition in the corresponding Gister frame. Thus, there is a one-to-one correspondence between the propositions in a frame and the fuzzy sets in the corresponding fuzzy domain. The fuzzy set for a given measurement-proposition pair is used to describe the mapping from the measurement to the proposition. This association between measurements, Gister frames, and fuzzy sets is defined by the user and placed in an input file.

The mass for a particular proposition in a frame is set as equal to the membership value of the measurement in the corresponding fuzzy set. The names of the input frames, the names of the input propositions, and the associated masses are stored in the frame-oriented data structure *analysis_input*, using the function **init_prop_mass()**. This function and *analysis_input* are discussed in Subsection 2.4.2.

2.4.2 Gister Analysis Module

The Gister Analysis Module is invoked at each epoch processing instant. It performs evidential reasoning analysis to draw conclusions about the environment. For flexibility, RTNP allows the contents of the Gister Analysis Module to be specified by the user. This module can be created either by writing a C function or by using the LISP-based Gister analysis tool.

The input to this module is the frame-oriented data structure *analysis_input*, which gives the masses of the propositions of the input frames of discernment (FODs). The Gister Analysis Module uses *analysis_input* to perform evidential reasoning, and draws conclusions about the environment in terms of the support and plausibility of various output propositions. These support and plausibility measures are placed in the data structure *analysis_output*, and are used by the Gister Output Summarizer Module to compute summarized estimates of the network environment.

Both *analysis_input* and *analysis_output* share the same data structure: both are arrays of elements, one for each frame, with each element having two fields. The first field contains the name of the input frame, and the second contains a pointer to a mass function that includes a list of input propositions and their associated mass, support, and plausibility (proposition-mass pairs). The support and the plausibility fields for *analysis_input* are assigned the value -1. Once all the elements of *analysis_input* are properly loaded, the Gister Analysis Module is called. For ease of coding, *analysis_input* can be loaded via the function call **init_prop_mass()**. The arguments required for **init_prop_mass()** are the name of an input frame, a pointer to an input proposition, and its associated mass.

The output of the Gister Analysis Module is stored in the data structure *analysis_output*. This data structure can be accessed via calls to the functions **get_prop_support()** and **get_prop_plausibility()**, which return the support and plausibility, respectively, of the given output proposition. The arguments required for these functions are the name of the output frame and a pointer to an output proposition.

2.4.3 Gister Output Summarizer Module

The Gister Output Summarizer Module is invoked at each epoch processing instant to summarize the support and plausibility measures for various propositions, obtained from the Gister Analysis Module, into a form that can be used by the HLNC Module to compute various protocol parameters. Flexibility is provided to output the support and plausibility measures unchanged, since they may also be used as input to the HLNC Module.

This module uses *output_fuzzy_set* and *output_fuzzy_domain* objects, as described in Subsection 2.5. The definitions of these objects are read from a file and stored as data structures before the simulation. Each *output_fuzzy_domain* is associated with a Gister frame whose name is also read in from the file. The number of propositions in a frame is equal to the number of fuzzy sets in the associated fuzzy domain. Thus, there is a one-to-one correspondence between a proposition in a frame and the corresponding fuzzy set in the associated fuzzy domain object. A new data structure is populated with Gister output frame names and the corresponding *output_fuzzy_domain* objects. This data structure exhibits one-to-one correspondence between a Gister frame name and its corresponding fuzzy domain object.

This module summarizes the support and plausibility of the various propositions in each output frame into a single number, which is taken to be an environment estimate. Two approaches are devised. In one approach, the summarization of an output frame is represented by the weighted average of the centroids of the fuzzy sets in the associated *output_fuzzy_domain* object, weighted by the supports of the corresponding propositions. In the other approach, the summarization of an output frame is represented by the weighted average of the centroids of the fuzzy sets in the associated *output_fuzzy_domain* object, weighted by the average of the support and plausibility of the corresponding propositions.

2.5 FUZZY LOGIC HLNC MODULE

The Fuzzy Logic HLNC Module is invoked at each epoch processing instant to dynamically control one or more STIP3 control parameters in response to changes in measurements and estimates of the environment. The output of this module can be any subset of the STIP3 control parameters. In experiments, we used the STIP3 parameters listed in Subsection 2.1.

Theoretically, the input variables of this module can be any measurements or estimates from the Neural-Network EE Module, the Gister EE Module, and/or the Measurements Module. However, for experimentation, we selected the following measurements and estimates as possible inputs:

- *ete_jamming_amt*, an estimate of the amount of end-to-end jamming averaged over all destinations (from the End-to-End Jamming Estimator Module).

- *ete_jamming_period*, an estimate of the period (1/rate) of end-to-end jamming averaged over all destinations (from the End-to-End Jamming Estimator Module).

- *congestion*, a measurement of local congestion (from the Measurements Module).

- *oscillation_period* = 1/*osc_degree*, where *osc_degree* is a measure of the maximum frequency at which the primary link for some destination is changing (from the Measurements Module).

- *traffic_burstiness*, a measurement of the burstiness of the local arrival traffic (from the Measurements Module).

- *jamming_traffic_correlation*, a measurement of the correlation between the amount of jamming and the amount of flow on the outgoing links (from the Measurements Module).

The Fuzzy Logic HLNC Module follows an objected-oriented design. Types of objects in the HLNC include *input_fuzzy_set*, *output_fuzzy_set*, *input_fuzzy_domain*, *output_fuzzy_domain*, and *fuzzy_rule*. Important functions in the Fuzzy Logic HLNC Module include the parsing of fuzzy sets, fuzzy domains, and fuzzy rules; the evaluation of fuzzy membership values; and the derivation of output variable values. These objects and functions are discussed in this section. In order to allow experimentation with different fuzzy rules, the user can define the above objects in an input file.

input_fuzzy_set. This object uses five floating-point numbers to describe the shape of an input fuzzy set. The fuzzy set has a trapezoid shape, with the first four floating-point numbers giving the x coordinates of the lower left, upper left, upper right, and lower right corners of the trapezoid, respectively, and the last floating-point number giving the y coordinate of the top of the trapezoid. The y coordinate of the base of the trapezoid is assumed to be 0.0. Objects of type *input_fuzzy_set* are defined only within objects of type *input_fuzzy_domain*. An example of an *input_fuzzy_set* is given in Figure 2.5 and is defined as follows:

small 0.0 2.0 6.0 8.0 1.0

output_fuzzy_set. This object uses two floating-point values to describe a fuzzy set for an output variable. These two floating-point values specify the centroid and the area of the fuzzy set. (The exact shape of the output fuzzy set need not be given, since we use the centroid method for defuzzification.) Objects of type *output_fuzzy_set* are defined only within objects of type *output_fuzzy_domain*. An example of an *output_fuzzy_set* is

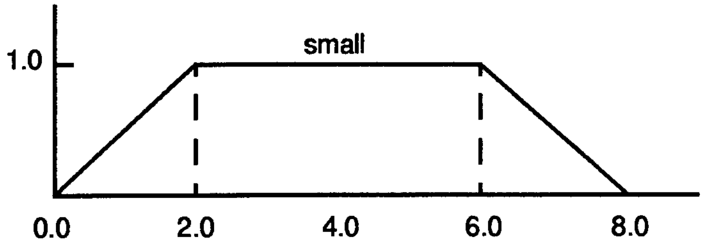

Figure 2.5. Example of an Input Fuzzy Set

large 0.75 5.0

input_fuzzy_domain. This object defines a fuzzy domain for an input variable. The definition of an *input_fuzzy_domain* includes the associated input variable name, the range of the domain, and a group of *input_fuzzy_sets.* An example of an *input_fuzzy_domain* is

input_fuzzy_domain jamming_rate_domain jam_rate 0.0 100.0

(small 0.0 2.0 6.0 8.0 1.0 large 0.0 10.0 15.0 20.0 5.0)

output_fuzzy_domain. This object defines a domain for an output variable. The definition of an *output_fuzzy_domain* includes the associated output variable name, the range of the domain, and a selected group of *output_fuzzy_set.* An example of an *output_fuzzy_domain* is

output_fuzzy_domain spr spread 0.0 1.0 (small 0.5 3.0 large 0.75 5.0)

fuzzy_rule. A fuzzy rule is of the form "If A then B," where the antecedent A is the conjunction of one or more input fuzzy sets, each corresponding to a different input fuzzy domain, and B is an output fuzzy set corresponding to some output fuzzy domain. Each fuzzy rule has an associated weight, which is applied when the rule is invoked. All fuzzy sets contained in a given antecedent are assumed to be chained together with the "AND" operator. The following example shows a rule with weight 1.0; the rule says that if the input variable "jamming rate" is small and the input variable "oscillation degree" is large, then the output variable spread should be large:

fuzzy_rule 1.0 (jamming_rate_domain small oscillation_degree_domain large)
(spread large)

Applying a rule requires evaluating the degree to which its antecedent is true. First, each input fuzzy set in the antecedent is evaluated by finding the membership value of the input variable with respect to the input fuzzy set. The membership value of an antecedent is then taken to be the minimum of these values. This mimimum value is used to compute, for each rule r, two terms X_r and Y_r as follows, where W_r is the weight of the rule, M_r is the (minimum) membership value of the antecedent of the rule, A_r is the area of the consequent fuzzy set of the rule, and C_r is the centroid of the consequent fuzzy set of the rule:

$$X_r = W_r M_r A_r C_r \tag{2.3}$$

$$Y_r = W_r M_r A_r \quad . \tag{2.4}$$

In the above manner, all fuzzy rules that apply to a given output variable are evaluated. The value of the output variable is the sum of X_r over all rules, divided by the sum of Y_r over all rules, i.e.,

$$fuzzy\_output = \frac{\sum_r W_r M_r A_r C_r}{\sum W_r M_r A_r} \quad . \tag{2.5}$$

Note that the fuzzy output is a weighted average of the centroids of the output fuzzy sets, where the weights are $W_r M_r A_r$. This formula for the output is the result of applying a commonly used technique called product-correlation inference, followed by using the centroid (or center-of-area) method for defuzzification [Kosko 1992; Masters 1993].

3 RTNP SIMULATION SOFTWARE

The RTNP simulation software was developed by modifying and augmenting the software already developed in EDMUNDS for simulating STIP3. Both the STIP3 and RTNP software are written in C++, using NAPI+. NAPI+ provides an interface between the protocol-independent code (called the NAPI+ Environment Module) and the protocol-dependent code (all the other modules), as shown in Figure 3.1. NAPI+ was developed in the EDMUNDS project and is described in the EDMUNDS Final Report [Hight et al. 1993].

Instructions for using the RTNP simulation software are given in the Software User's Manual [Wong, Ogier, and Khan 1995]. Here we provide only a brief overview of how this software is used.

The RTNP simulation software reads a number of input files at the initialization of a simulation. These include the environment file, which specifies protocol and simulation parameters and options; the script file, which describes the possibly dynamic network topology; files that contain the neural network parameters; and files that describe the fuzzy sets and rules used by the HLNC. The Software User's Manual provides a description of these files and parameters, and also describes how to train the neural networks used by RTNP.

The main output of the RTNP simulation software is a log file, which can be used by a number of analysis tools to provide and plot performance data. These tools include plot_ete, netviz, p2, and p3, which were developed in EDMUNDS and are described in [Hight et al. 1993]. Netviz is a graphical network-display tool based on X Windows.

Two new tools, go.pl and paint.pl, were developed within the RTNP project. Go.pl is a Perl script developed to facilitate batch processing of large numbers of RTNP simulations, and paint.pl is another Perl script developed to summarize different simulation results by presenting them on a graphical plot. These tools are described in the Software User's manual.

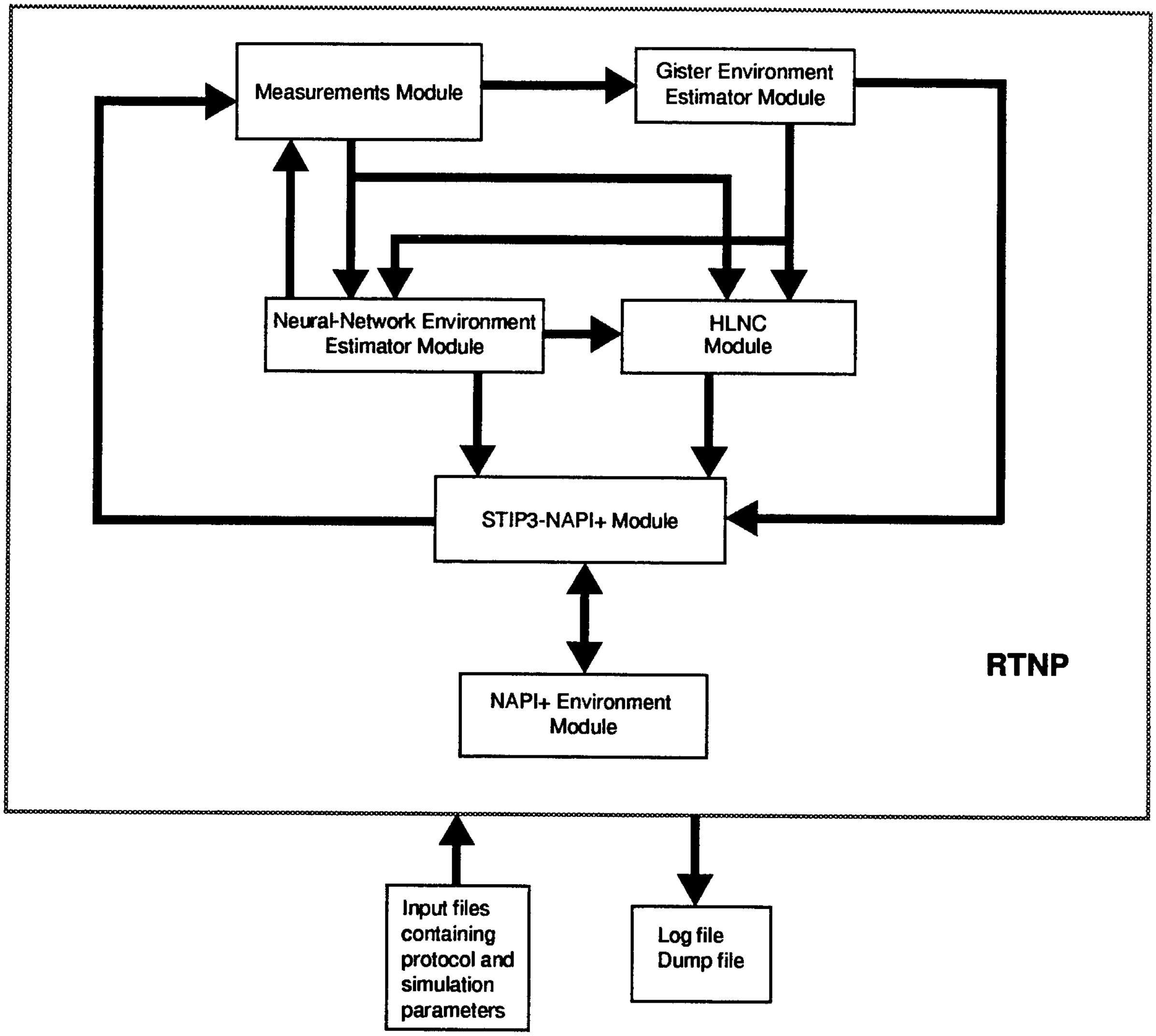
Measurements Module
Gister Environment Estimator Module
Neural-Network Environment Estimator Module
HLNC Module
STIP3-NAPI+ Module
NAPI+ Environment Module
RTNP
Input files containing protocol and simulation parameters
Log file
Dump file

Figure 3.1. RTNP Software Modules

4 SIMULATION TEST RESULTS

The Software Test Report [Khan, Ogier, Wong 1995b] describes the tests performed for RTNP and presents their results. This section presents a summary of those results. Three types of experiments were conducted: (1) tests evaluating the Neural-Network Link-State Predictor Module and Gister for predicting link states; (2) tests evaluating the Neural-Network Queuing Delay Predictor Module; and (3) tests evaluating the Fuzzy Logic HLNC Module.

4.1 NEURAL-NETWORK AND GISTER LINK-STATE PREDICTORS

Link-state probability predictions obtained using the Neural-Network Link-State Predictor Module and a Gister link-state predictor were compared with predictions obtained using the simple STIP3 link-state probability predictor, described in Hight et al. [1993]. Two types of tests were performed.

4.1.1 Tests on a Single Link

In each of these tests on a ten-node network, all links in the network were jammed using the same jamming model and jamming parameters. The neural network was trained and tested on a single link. Three different jamming models (Markov, Nth-order Markov, and periodic) were used, with three different jamming rates (slow, medium, fast), for a total of nine tests.

The average mean-squared error for the neural-network predictor was about half that of STIP3 and less than half that of Gister, implying that the neural-network predictor is much more accurate than the other two methods.

4.1.2 Tests on Multiple Links

In each of these tests on a ten-node network, nine links were jammed using the nine different jamming models/rates. The neural network was trained and tested on these nine links, on three additional links that were not jammed (so that the neural network can learn to identify an unjammed link), and on the reverse of each of these links (24 links in all).

The average mean-squared error for STIP3 was about 1.6 times that of the nerual network, implying again that the neural-network predictor is much more accurate than the STIP3 predictor. (Gister was not tested in this experiment.)

4.2 NEURAL-NETWORK QUEUING DELAY PREDICTOR MODULE

Queuing delay predictions obtained using this module were compared with those obtained using the STIP3 queuing delay predictor heuristic. Two types of tests were performed.

4.2.1 Tests on a Single Destination Queue

In these tests on a ten-node network, queuing delay predictions obtained by the two methods were compared for one destination queue at one node. Six different traffic models were used, representing different loads and different degrees of burstiness, and medium-rate Markov jamming was used on all outgoing links of the queue that was observed.

The average mean-squared error for STIP3 was about 3.2 times that of the neural network, implying that the neural network is a much better predictor.

4.2.2 Tests on Multiple Destination Queues

These tests used a network with four nodes and four bidirectional links, forming a square topology. A single bursty traffic stream was used in which the source and destination were opposite nodes on the square. The two links going into the single destination were jammed independently using medium-rate Markov jamming. The predicted queuing delay (whether obtained using the neural network or STIP3) was used in the STIP3 routing computation. Queuing delay predictions obtained by the two methods were compared for the three queues at the three nondestination nodes. We also compared the end-to-end delay and throughput.

The average mean-squared error for STIP3 (in predicting the queuing delay) was 3.66 times that of the neural network, again showing that the neural network is a much better predictor. In addition, the throughput was 55% higher for the neural network method, while the end-to-end delays for the two methods were about the same. This shows that the improved queuing delay predictions of the neural network, when used for making routing decisions, can greatly improve end-to-end performance.

4.3 FUZZY LOGIC HLNC MODULE

For these tests, the performance of the RTNP protocol, including the Fuzzy Logic HLNC Module, was compared with the performance of STIP3. Neural networks were not used in these experiments. Two fuzzy rule bases for the HLNC were tested, one for controlling the *spread* parameter and the other for controlling the *link_bias* parameter.

4.3.1 Tests for the HLNC that Controls Spread

In these tests the HLNC used a fuzzy rule base that controlled the STIP3 parameter *spread* as a function of *ete_jamming_amt* and *ete_jamming_period.* Four jamming scenarios were used: no jamming, slow jamming, medium jamming, and fast jamming. For each jamming scenario, one test was performed with the HLNC, and three tests were performed with STIP3, each using a different setting for *spread.* Average end-to-end packet delay was measured.

For all jamming models, RTNP with the HLNC performed at least as well as STIP3 with any of the settings for *spread*, and for any single setting for *spread*, there is at least one jamming model for which RTNP performed significantly better than STIP3. Therefore, the Fuzzy Logic HLNC Module succeeds in selecting the best value for the spread parameter for all of the jamming models.

4.3.2 Tests for the HLNC that Controls Link_bias

In these tests the HLNC used a fuzzy rule base that controlled the STIP3 parameter *link_bias* as a function of *ete_jamming_amt* and *ete_jamming_period.* Four jamming scenarios were used: no jamming, slow jamming, medium jamming, and fast jamming. For each jamming scenario, one test was performed with the HLNC, and three tests were performed with STIP3, each using a different setting for *link_bias.* Average end-to-end packet delay was measured.

For all jamming models, RTNP with the HLNC performed at least as well as STIP3 with any of the settings for *link_bias.* However, in all jamming models, STIP3 with *link_bias* set to 0.5 performed as well as RTNP in terms of the average delay. Thus, in this experiment, the optimal setting for *link_bias* does not depend on the jamming period or on whether jamming is present.

5 CONCLUSIONS

The most significant conclusion to be drawn from the experiments is that our neural network methods are very effective at predicting link-state probabilities and queuing delays, and are much more effective than either Gister or STIP3. Moreover, the superior predicting ability of neural networks can result in dramatically improved network performance.

The test results for the Fuzzy Logic HLNC Module show that using fuzzy logic can improve network performance by automatically selecting the best parameter settings. However, the use of fuzzy logic is limited by the ability of the human expert to find rules for controlling the protocol parameters as a function of network measurements and estimates. It is difficult to find effective rules for even a single scenario. It is even more difficult to find rules that work for multiple scenarios, since a rule that works for one scenario may not be effective for another.

An effective rule was found for controlling the spread parameter as a function of the amount and rate of end-to-end jamming, but we found that the optimal choice for the *link_bias* parameter did not depend on the jamming estimates. In addition, we tried unsuccessfully to find relationships between the STIP3 parameters *epsilon*, *alpha_inc*, and *short_update_period*, and the RTNP environment estimates *corr_jam_traffic* (which estimates the correlation between jamming and traffic), *congestion*, *traffic_burstiness*, and *osc_period* (which measures the degree of routing oscillations). However, our search for such rules was limited, and further research may result in the discovery of new effective rules. We note that fuzzy logic has been shown to be useful in some applications.

The application of Gister in network control is limited by its ability to learn, its complexity, and its requirement that different information sources be independent. However, since our effort was limited, this does not prove that Gister is ineffective for network control. We note that Gister has been shown to be useful in other applications, such as situation assessment.

A feature of neural nets that makes them effective for network control is their ability to learn from training data, without relying on human expertise which may not exist. Although we used neural nets only for link-state and queuing delay prediction, they are also a promising alternative to fuzzy logic in the HLNC. In fact, a hybrid of fuzzy logic and neural nets can be used. In addition, neural nets may be useful for estimating, detecting, or recognizing other things such as anomalous routing behavior and network intrusions.

Some suggestions for future work are as follows:

- Apply neural nets to HLNC and to recognizing or predicting other adverse conditions, including network intrusions.
- Apply the ideas of RTNP to other protocols besides STIP3, including ATM protocols.
- Apply the ideas to specific networks such as multiband multimode radio networks.
- Apply the ideas of RTNP to network management.
- Consider case-based reasoning, which is another promising AI method, in addition to neural networks.

6 REFERENCES

Haykin, S. 1994. *Neural Networks—A Comprehensive Foundation*, Macmillan College Publishing Company, New York.

Hight, J., E. Costa, D. Lee, R.G. Ogier, and J.S. Wong. 1993. *Evaluation and Development of Multimedia Networks in Dynamic Stress (EDMUNDS)*, ITAD-8558-FR-93-277, SRI International, Menlo Park, California.

Khan, I., R.G. Ogier, and J.S. Wong. 1995a. *Software Design Document for the Robust Transfiguring Network Protocol*, ITAD-3302-DD-95-217, SRI International, Menlo Park, California (November).

Khan, I., R.G. Ogier, and J.S. Wong. 1995b. *Software Test Report for the Robust Transfiguring Network Protocol*, ITAD-3302-MS-95-166, SRI International, Menlo Park, California (August).

Kosko, B. 1992. *Neural Networks and Fuzzy Systems*, Prentice Hall, Englewood Cliffs, New Jersey.

Lowrance, J.D. 1993. *Evidential Reasoning with Gister-CL—A Manual*, Technical Report, SRI International, Menlo Park, California.

Masters, T. 1993. *Practical Neural Network Recipes in C++*, Academic Press, New York.

Wong, J.S., R.G. Ogier, and I. Khan. 1995. *Software User's Manual for the Robust Transfiguring Network Protocol*, SRI International, Menlo Park, California (April).

*U.S. GOVERNMENT PRINTING OFFICE: 1996-710-126-20212

MISSION
OF
ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.